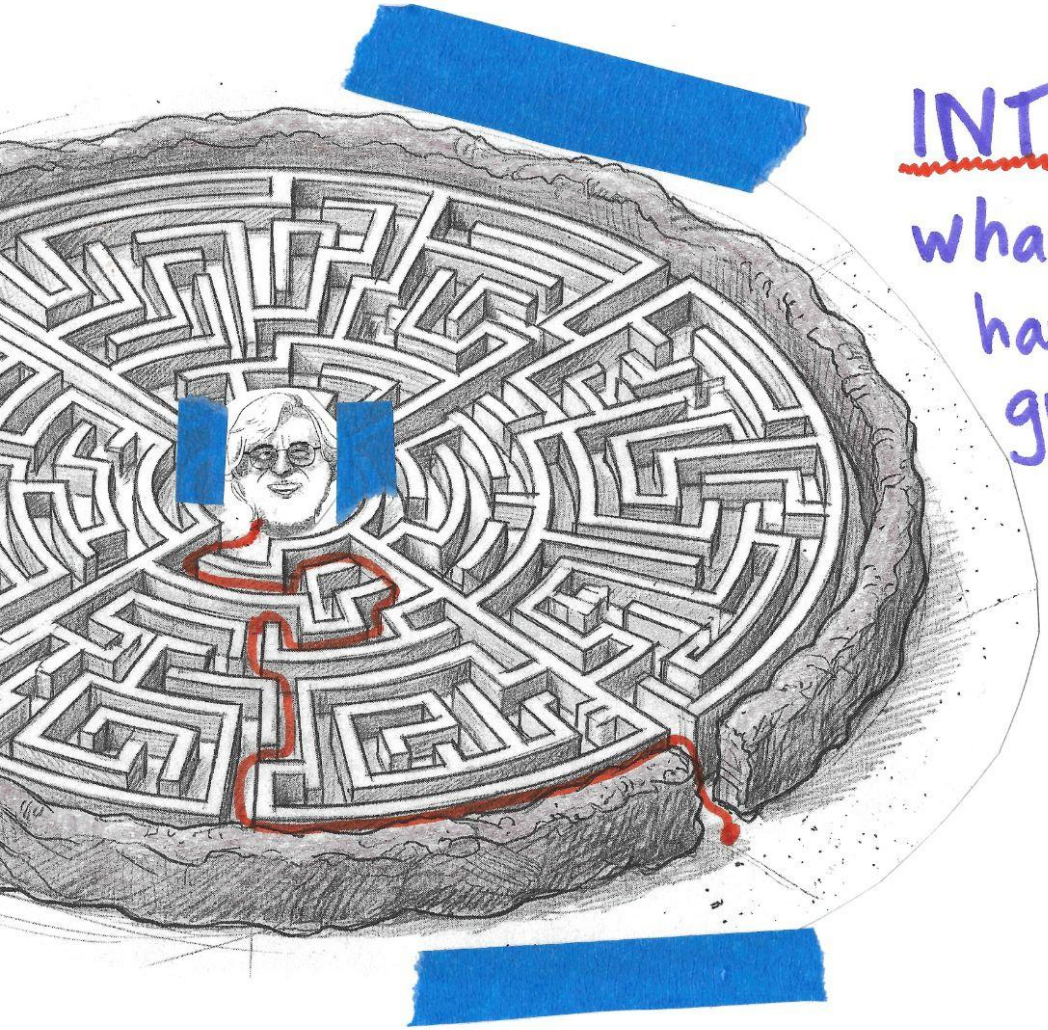


# INTO THE LABYRINTH: what distributed systems have taught us about guardrails

Akshay Shah
@akshayshah

Nov 16, 2026

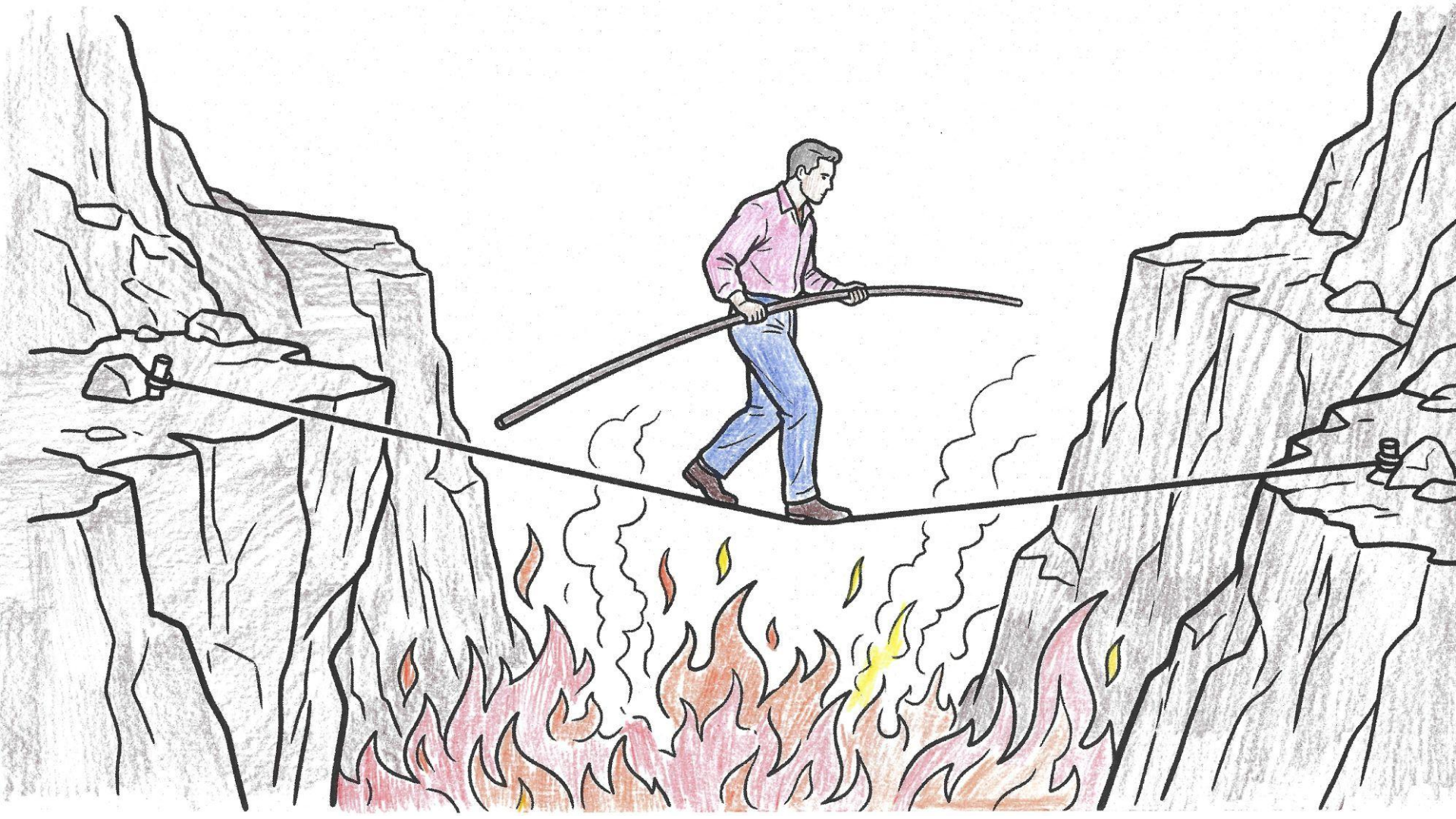
# AKSHAY SHAH

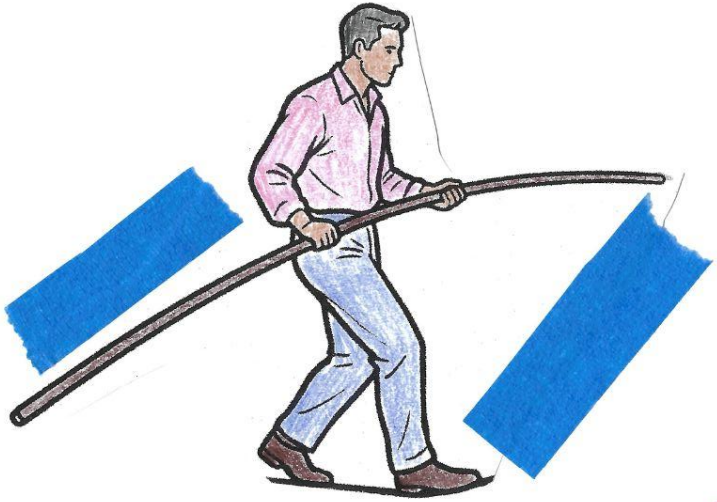
Infra + platform eng:

- Uber
- MSFT
- my silly startup
- other silly startups

Now at Antithesis!





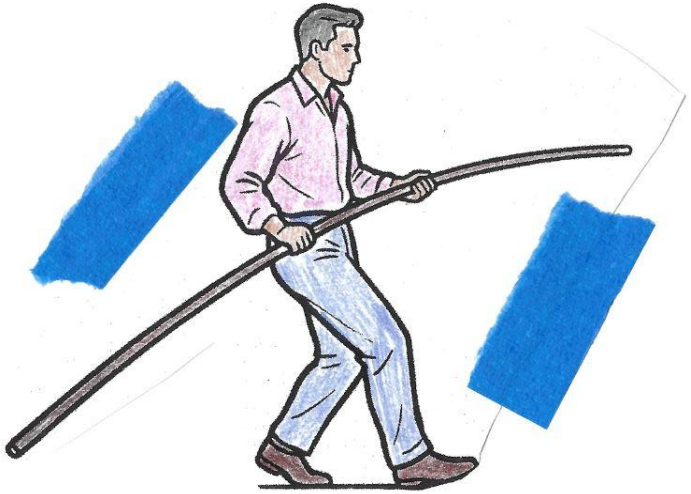


## TESTING

unit tests
integration tests
observability
chaos testing
static analysis

## THINKING

architecture
code review
style guides
formal methods



THINKING

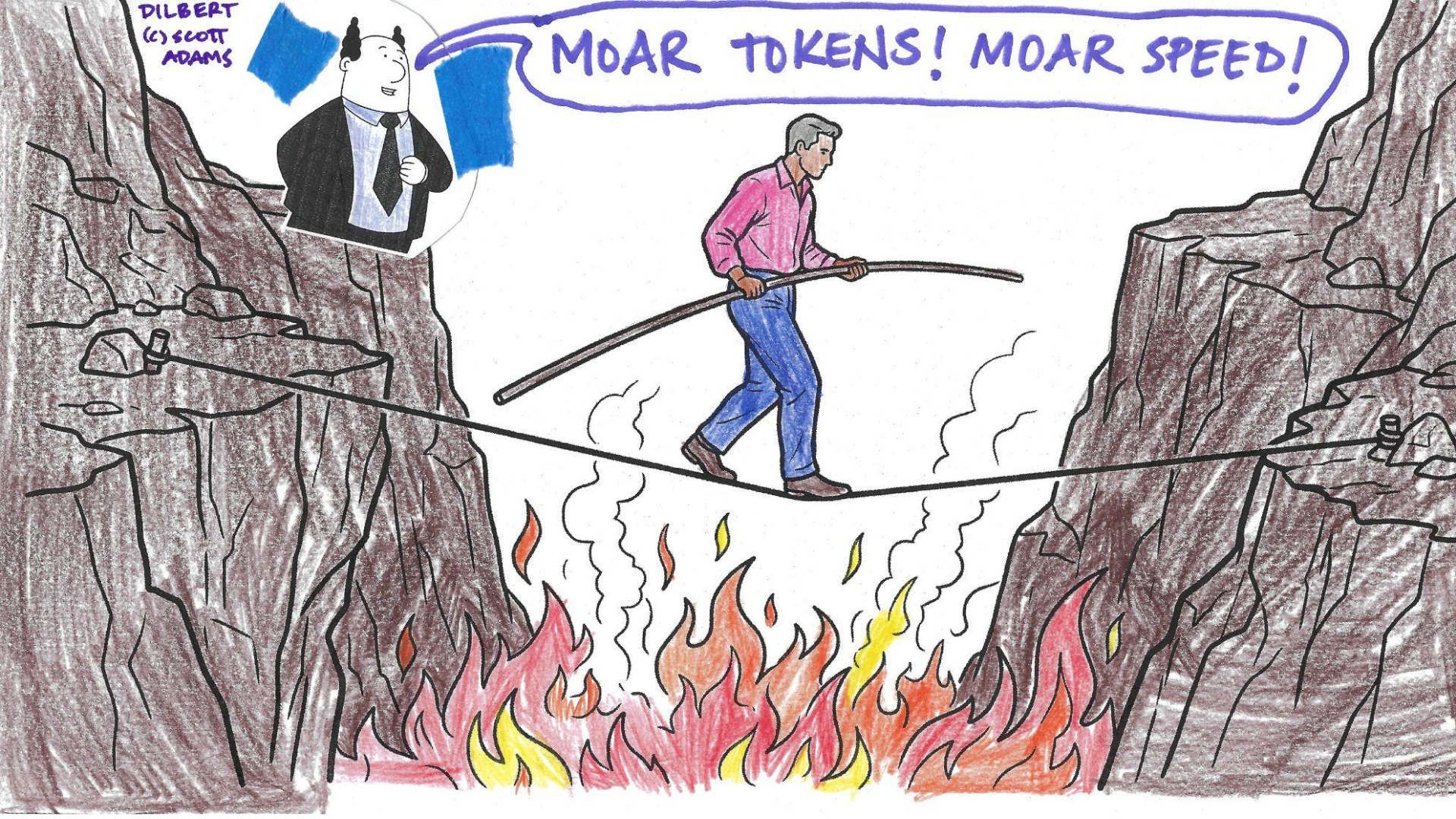
anticipate bugs

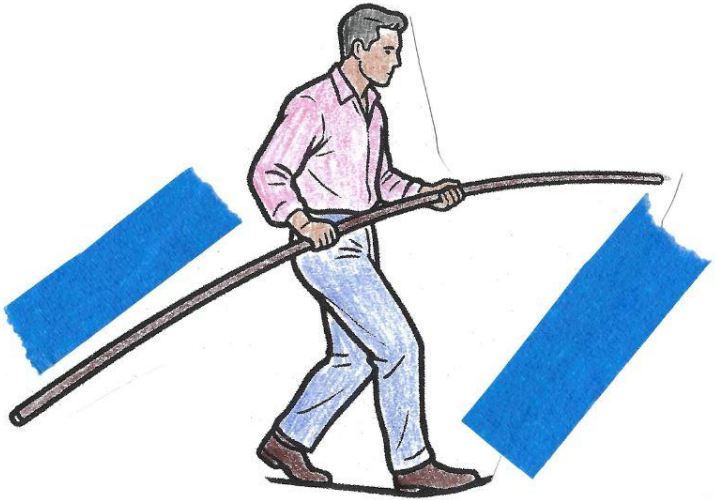
update design

strengthen tests

TESTING

catch known bugs


DILBERT
(c) SCOTT ADAMS
MOAR TOKENS! MOAR SPEED!



TESTING

tests

n tests

agents?

ous

chaos

static ana

THINKING

architectu

code r

stu

agents?

ods

# JEPSEN

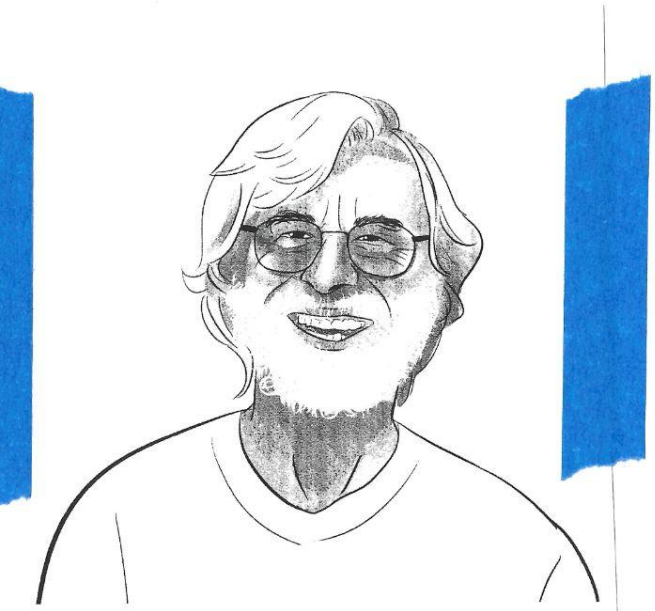
# ParticleDB 1.1.4

Kyle Kingsbury

2026-10-31

We found 96 issues with ParticleDB, which we present in several categories below. All but three of these issues occurred in healthy, single-node instances of ParticleDB. Busy readers may wish to jump to the Discussion section, which summarizes our findings.

However, our work on ParticleDB (and other databases) suggests there are serious tradeoffs to relying on LLMs. Quality suffered: ParticleDB was a fractal of correctness errors. LLM tooling repeatedly claimed that bugs had been fixed when, in fact, they had not. ParticleDB also seemed more difficult to understand than typical databases. As with most Jepsen analyses, Orkes engineers offered detailed explanations of the causes and resolutions of the bugs we found in Conductor. However, Orkes seemed unable to explain the causes or resolutions of nearly every issue we found in ParticleDB. New builds simply materialized, sans changelogs, each hopefully containing a slew of improvements. Why did ParticleDB sometimes turn strings into integers? Why did it stop in 1.1.18.11.rc-1? Perhaps no one knows. With LLMs, perhaps no one needs to.



"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable."

-Leslie Lamport

(patron saint of doomed causes)

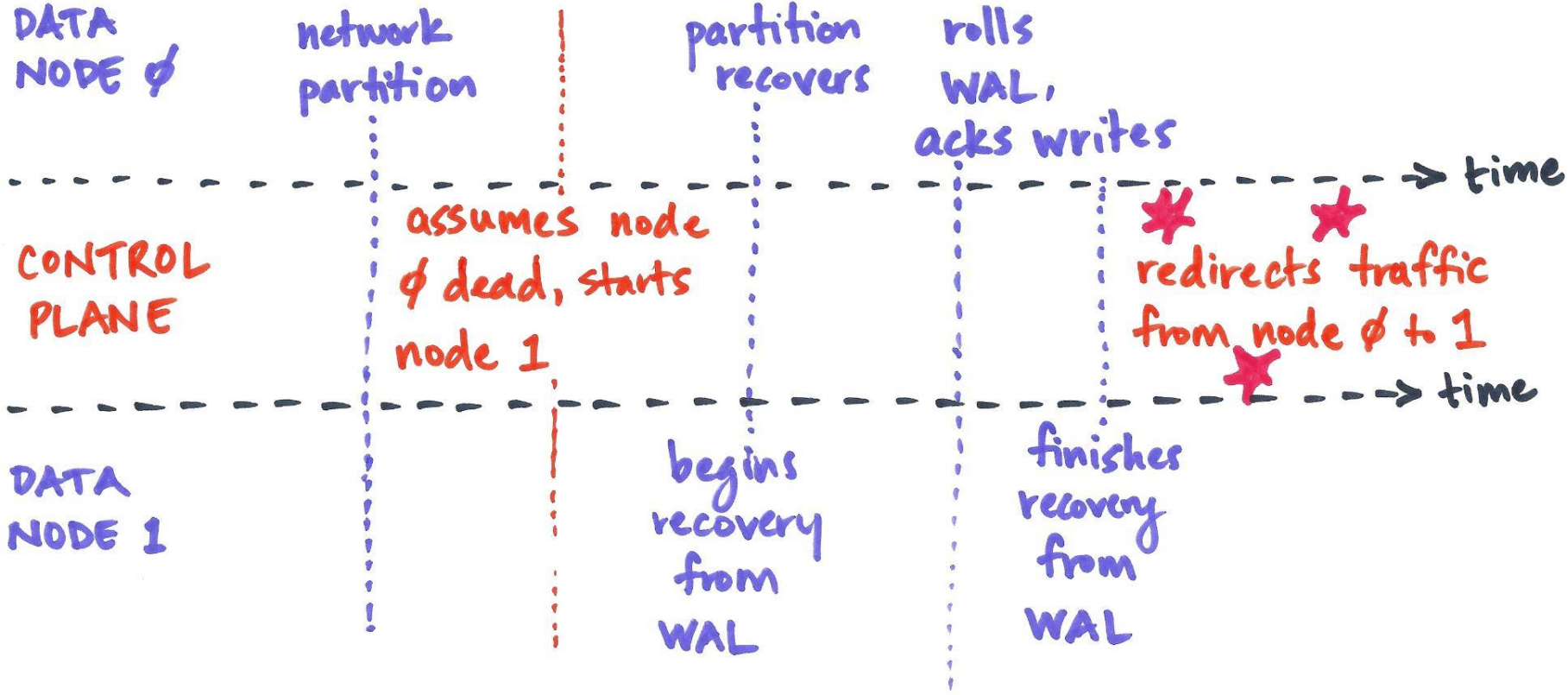
# Simple Testing Can Prevent Most Critical Failures: An Analysis of Production Failures in Distributed Data-Intensive Systems

**Ding Yuan, Yu Luo, Xin Zhuang, Guilherme Renna Rodrigues, Xu Zhao, Yongle Zhang, Pranay U. Jain, and Michael Stumm,** ***University of Toronto***

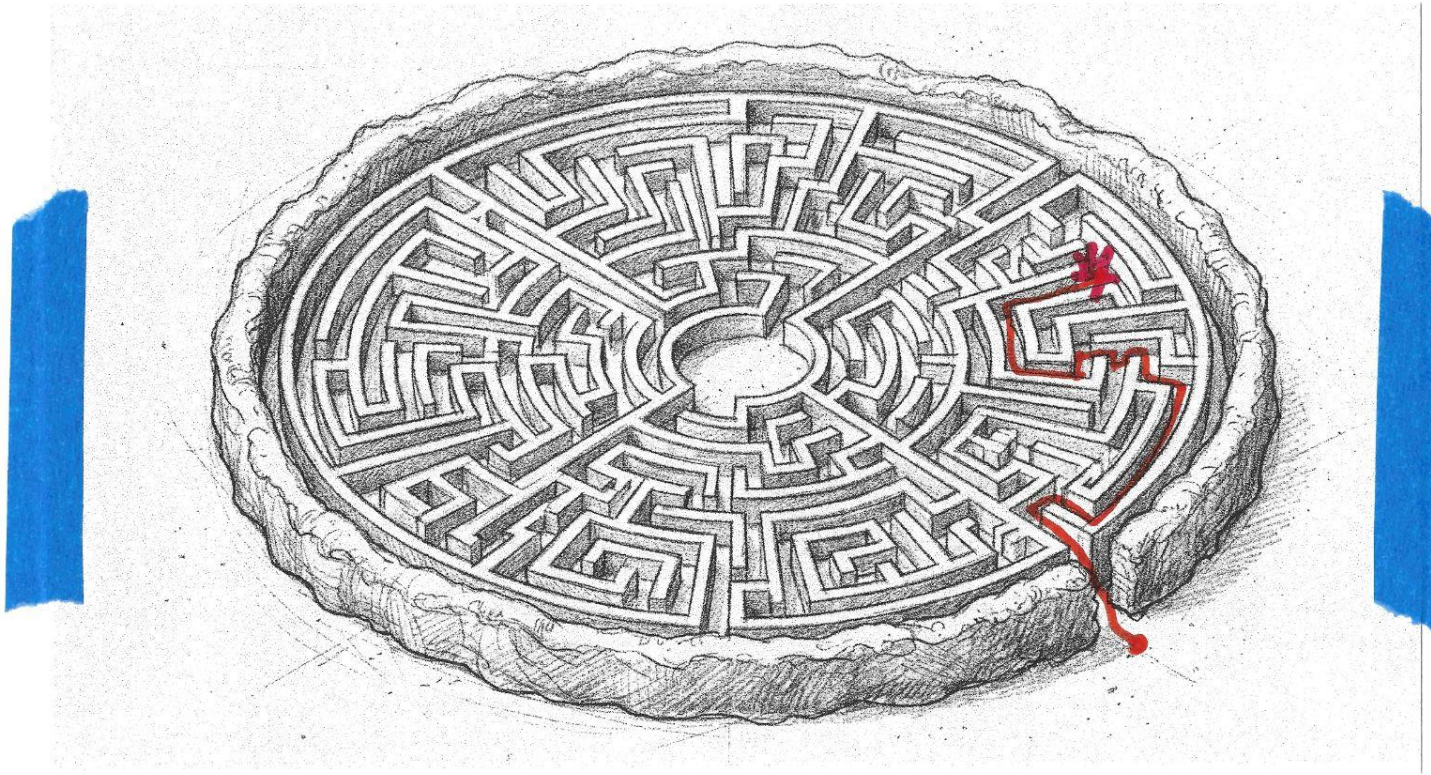
| Software | lang. | failures | | |
|---|---|---|---|---|
| | | total | sampled | catastrophic |
| Cassandra | Java | 3,923 | 40 | 2 |
| HBase | Java | 5,804 | 41 | 21 |
| HDFS | Java | 2,828 | 41 | 9 |
| MapReduce | Java | 3,469 | 38 | 8 |
| Redis | C | 1,192 | 38 | 8 |
| Total | – | 17,216 | 198 | 48 |

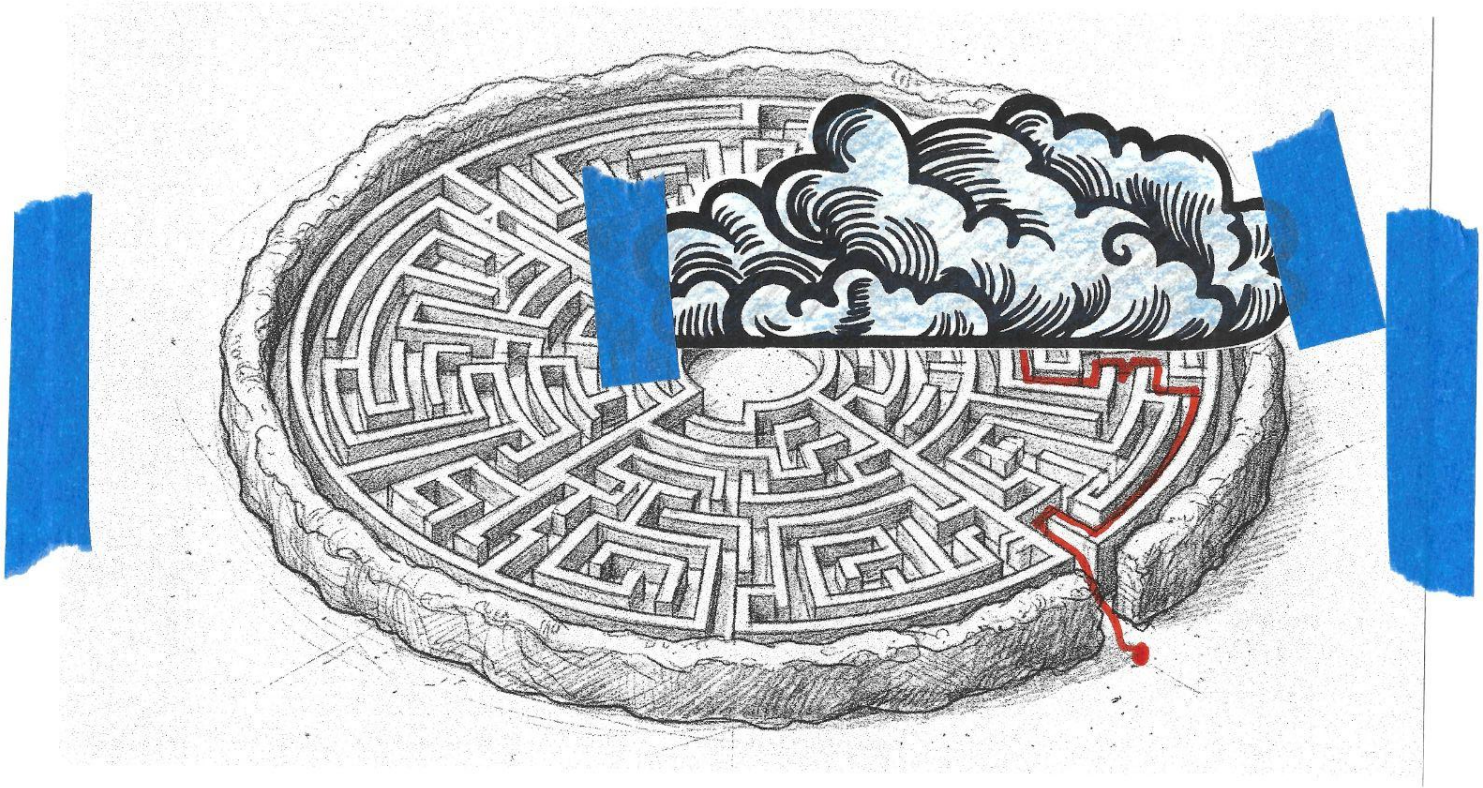
Table 1: Number of reported and sampled failures for the systems we studied, and the catastrophic ones from the sample set.

**Finding 9** *A majority of the production failures (77%) can be reproduced by a unit test.* (See Table 8.)


DATA NODE ∅
network partition
partition recovers
rolls WAL, acks writes
time
CONTROL PLANE
assumes node ∅ dead, starts node 1
redirects traffic from node ∅ to 1
time
DATA NODE 1
begins recovery from WAL
finishes recovery from WAL

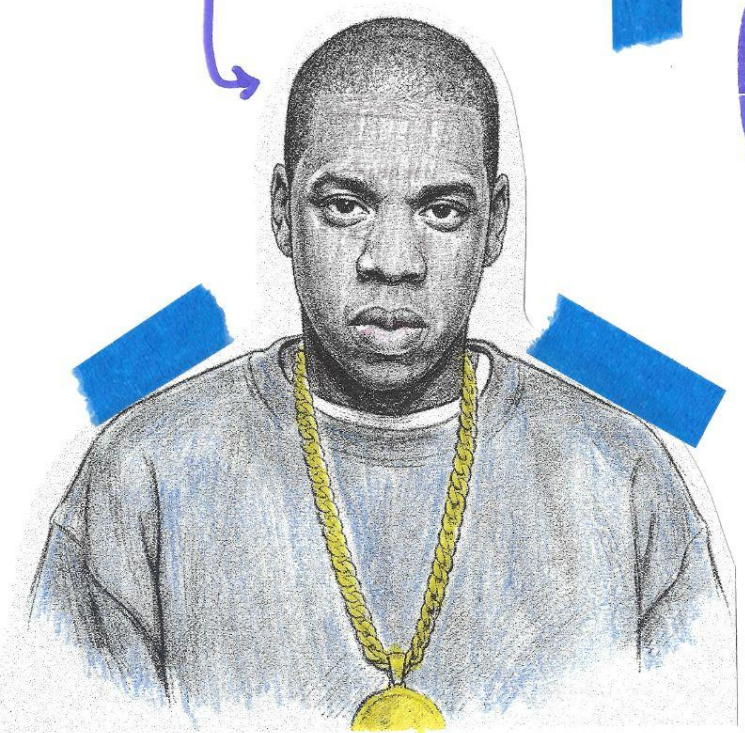
```
public void testLogRollAfterSplitStart {
    startMiniCluster(3); // okay...
    Leader.simulateUnreachableDataNode(0); // simulation?
    DataNodes.get(0).rollWAL(); // WTF :::
    for (i = 0; i < NUM_WRITES; i++)
        DataNodes.get(0).writer.append(...);
    Leader.reassignTrafficToDataNode(0, 1) // OMG kill me
    assertEquals(NUM_WRITES, DataNodes.get(1).countWrites())
}
```





| System | Date | Version |
|---|---|---|
| ParticleDB | 2026-10-31 | 1.1.4 |
| Orkes Conductor | 2026-10-20 | 5.3.2 |
| MariaDB Galera Cluster | 2026-03-16 | 12.1.2 |
| NATS | 2025-12-08 | 2.12.1 |
| Capela | 2025-08-07 | dda5892 |
| TigerBeetle | 2025-06-06 | 0.16.11 |
| Amazon RDS for PostgreSQL | 2025-04-29 | 17.4 |
| Bufstream | 2024-11-12 | 0.1.0 |
| jetcd | 2024-08-08 | 0.8.2 |
| Datomic | 2024-05-15 | 1.0.7075 |
| RavenDB | 2024-01-31 | 6.0.2 |
| MySQL | 2023-12-19 | 8.0.34 |
| Redpanda | 2022-04-29 | 21.10.1 |
| Radix DLT | 2022-02-05 | 1.0-beta.35.1 |
| Scylla | 2020-12-23 | 4.2-rc3 |
| Redis-Raft | 2020-06-23 | 1b3fbf6 |
| PostgreSQL | 2020-06-12 | 12.3 |
| MongoDB | 2020-05-15 | 4.2.6 |
| Dgraph | 2020-04-30 | 1.1.1 |
| etcd | 2020-01-30 | 3.4.3 |
| YugaByte DB | 2019-09-05 | 1.3.1 |
| TiDB | 2019-06-12 | 2.1.7 |
| YugaByte DB | 2019-03-26 | 1.1.9 |
| FaunaDB | 2019-03-05 | 2.5.4 |
| MongoDB | 2018-10-23 | 3.6.4 |
| Dgraph | 2018-08-23 | 1.0.2 |
| Aerospike | 2018-03-07 | 3.99.0.3 |
| Hazelcast | 2017-10-06 | 3.8.3 |
| Tendermint | 2017-09-05 | 0.10.2 |
| CockroachDB | 2017-02-16 | beta-20160829 |
| MongoDB | 2017-02-07 | 3.4.0-rc3 |
| VoltDB | 2016-07-12 | 6.3 |
| RethinkDB | 2016-01-22 | 2.2.3 |
| RethinkDB | 2016-01-04 | 2.1.5 |

Jay-Z, not John Hughes



# QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs


Koen Claessen
Chalmers University of Technology
koen@cs.chalmers.se

John Hughes
Chalmers University of Technology
rjmh@cs.chalmers.se


AWS calls this approach "lightweight" or "semi-formal" methods

also "generative" or "property-based" testing



```
def test_sorted():
    assert sorted([2, 1, 3]) == [1, 2, 3]
```

a little thin ...



```
def test_sorted():
    assert sorted([2, 1, 3]) == [1, 2, 3]
    assert sorted([]) == []
    assert sorted([1]) == [1]
```

getting better...



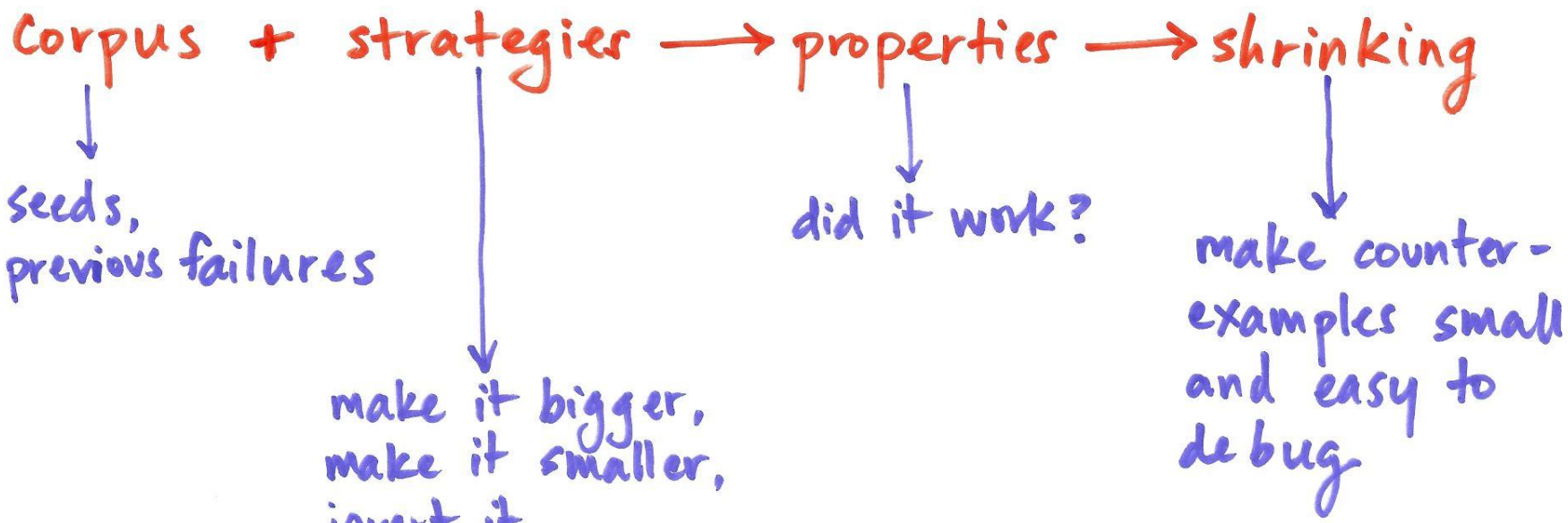
```
from collections import Counter
from hypothesis import given, strategies as st


@given(st.lists(st.integers()))
def test_sorted(xs):
    result = sorted(xs)
    for a, b in zip(result, result[1:]):
        assert a <= b
    assert Counter(result) == Counter(xs)
```

try all sorts of lists of integers, looking for cases where sorted(xs)...

isn't ascending...

or isn't a permutation of the input.

# PROPERTY-BASED TESTING IN A NUTSHELL



# AFLplusplus/include/config.h

```
276  #define INTERESTING_32                                          \
277    -2147483648LL,  /* Overflow signed 32-bit when decremented */ \
278        -100663046, /* Large negative number (endian-agnostic) */ \
279        -32769,     /* Overflow signed 16-bit                  */ \
280        32768,      /* Overflow signed 16-bit                  */ \
281        65535,      /* Overflow unsig 16-bit when incremented  */ \
282        65536,      /* Overflow unsig 16 bit                   */ \
283        100663045,  /* Large positive number (endian-agnostic) */ \
284        2147483647                  /* Overflow signed 32-bit when incremented */
285
```

# PROPERTY-BASED TESTING IN A NUTSHELL

corpus + strategies → properties → shrinking

[3] [3, 1] [] [1]

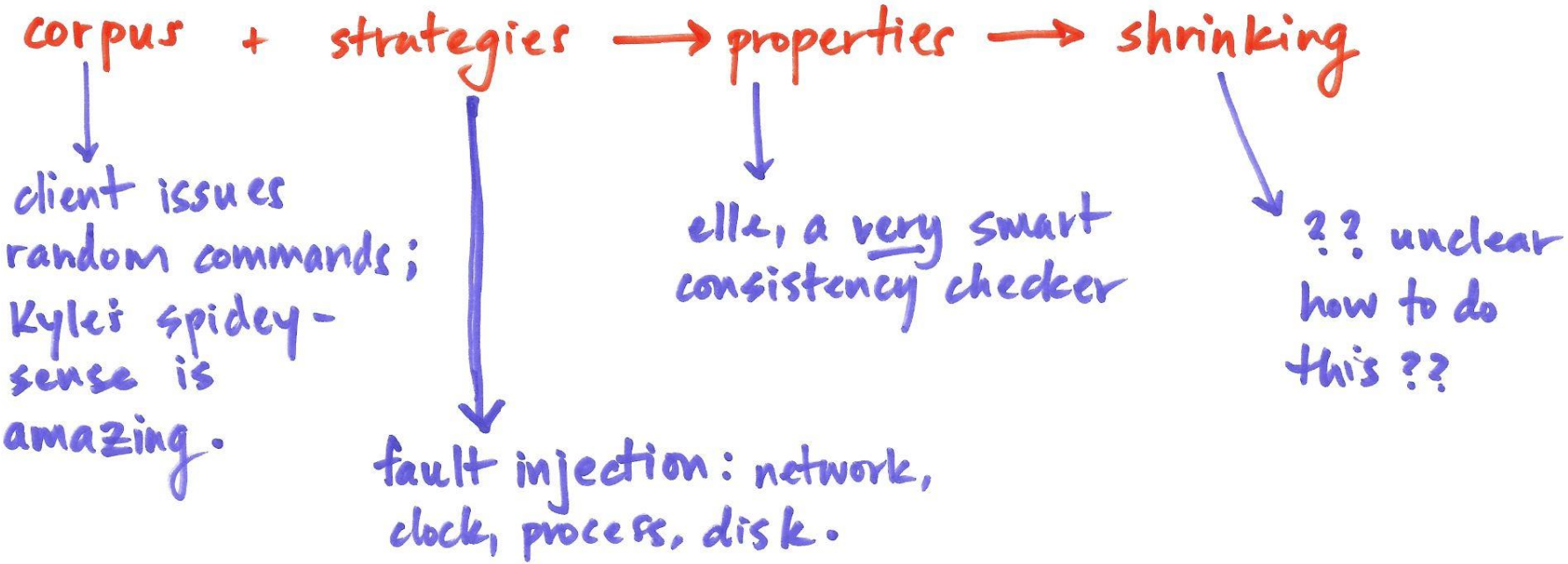
make it bigger!

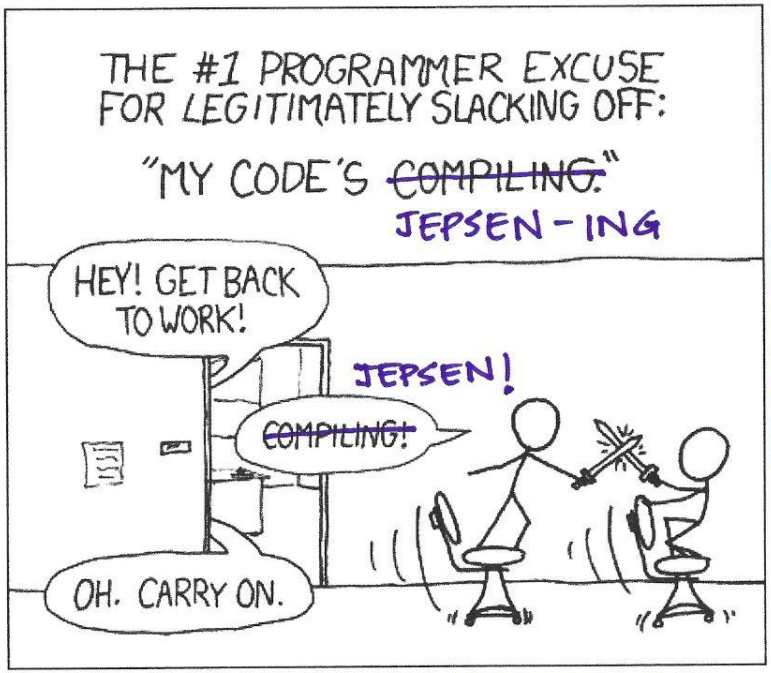
✓ ascending

✗ permutation

drop an element

# PROPERTY-BASED TESTING IN JEPSEN

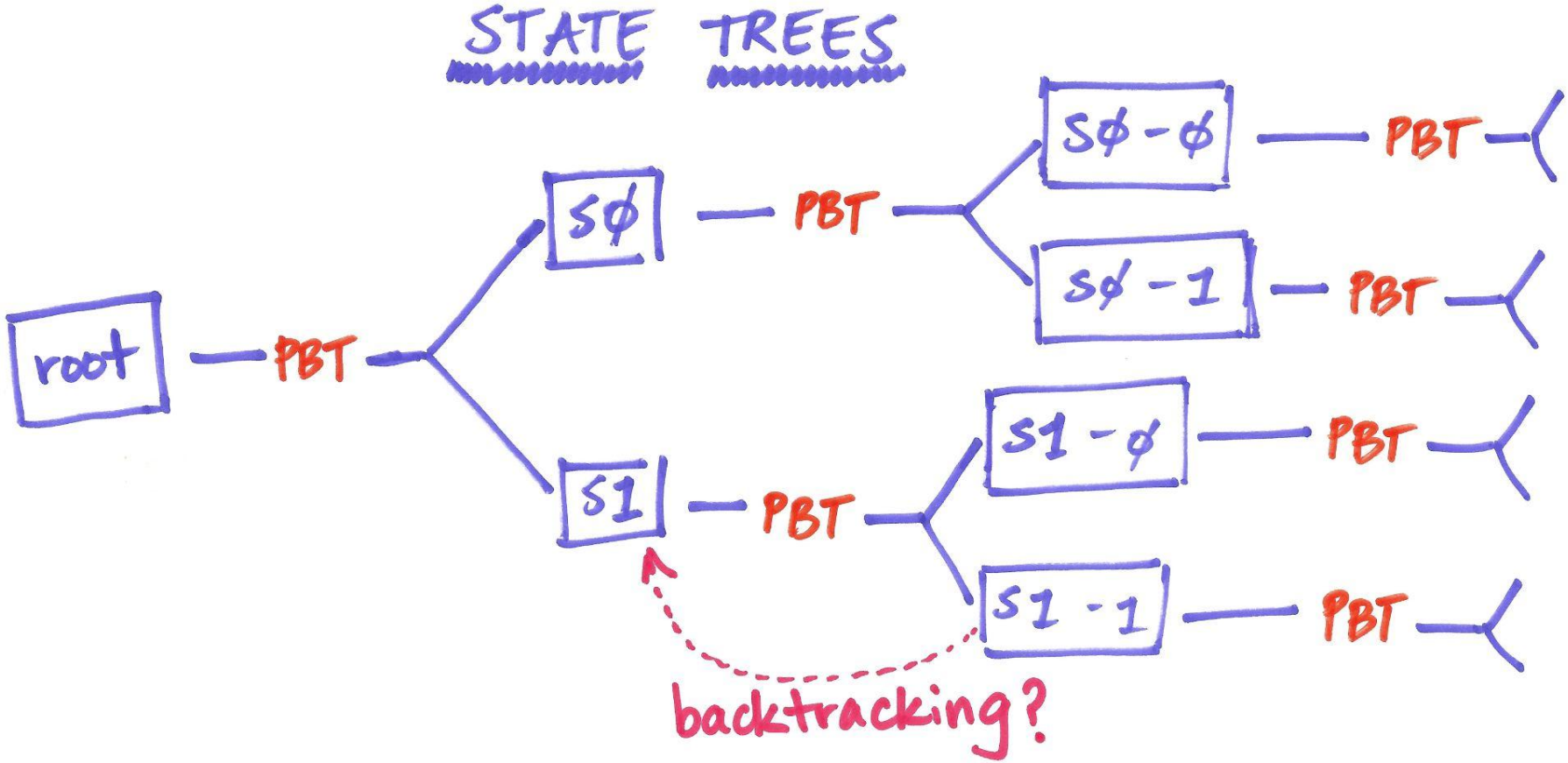



THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S ~~COMPILING.~~"
JEPSEN-ING
HEY! GET BACK TO WORK!
JEPSEN!
~~COMPILING!~~
OH. CARRY ON.
© XKCD #303

STATE

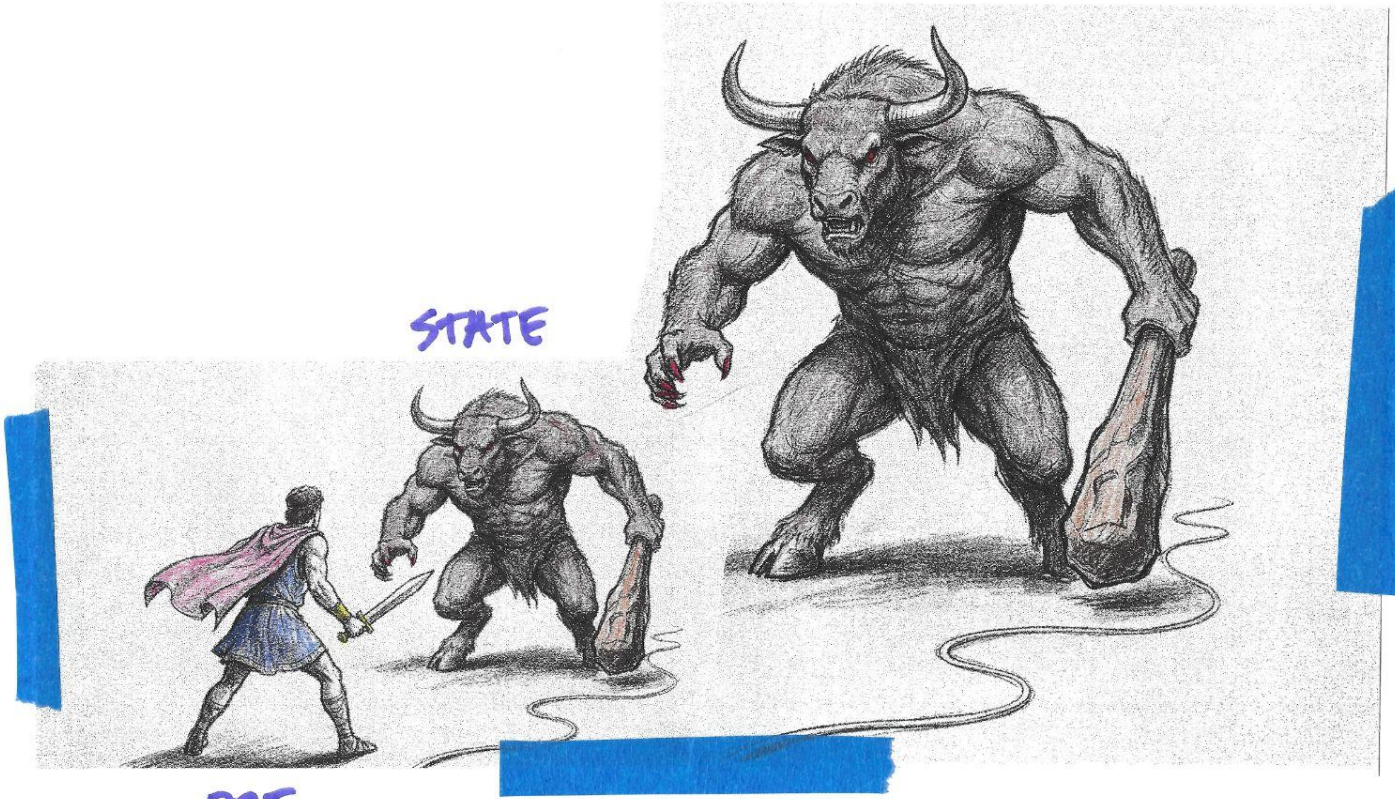


PBT


STATE TREES
root
PBT
sϕ
PBT
s1
PBT
sϕ-ϕ
PBT
sϕ-1
PBT
s1-ϕ
PBT
s1-1
PBT
backtracking?

NON-
DETERMINISM

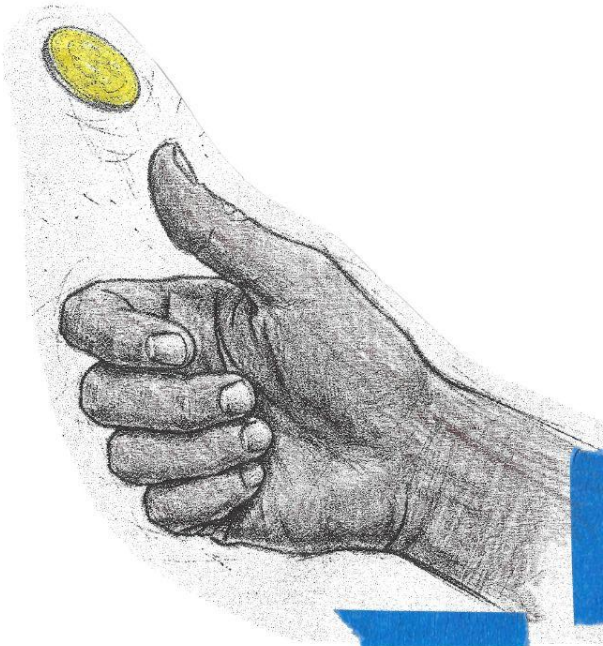
STATE



PBT

# NON-DETERMINISM IS EVERYWHERE

- system randomness
- performance
- thread scheduling
- network operations
- clock
- faulty hardware



# ELIMINATING NON-DETERMINISM

Two options:

(1) INTERNAL - dependency injection and mocks for all nondeterministic operations (e.g., Zig, Turmoil in Rust)

(2) EXTERNAL - run code in a deterministic VM (e.g., RewindVM, WASM, Antithesis)

# tigerbeetle / src / vsr / clock.zig

```
//! Cluster-wide synchronized clock, aggregating timing
//! information from all replicas.

pub fn init(
    allocator: std.mem.Allocator,
    time: Time,
    tracer: ?*Tracer,
    options: struct { ... },
) !Clock {
    var self = Clock{ .time = time };
    return self;
};
```

# tigerbeetle / src / stdx / time.zig

```
pub const Time = struct {
    context: *anyopaque,
    vtable: *const VTable,

    const VTable = struct { ... };

    pub fn monotonic(self: Time) Instant {
        return self.vtable.monotonic(self.context);
    }
    pub fn realtime(self: Time) InstantUnix {
        return self.vtable.realtime(self.context);
    }
};
```

# tigerbeetle/src/stdx/time.zig

```
/// Real Time backed by the operating system.
pub const TimeOS = struct {
    pub fn interface(self: *TimeOS) Time {
        return .{ ... };
    };
};

/// Simulated Time for testing.
pub const TimeSim = struct {
    pub fn interface(self: *TimeSim) Time {
        return .{ ... };
    };
};
```

# INTERNAL DETERMINISM: PROS AND CONS

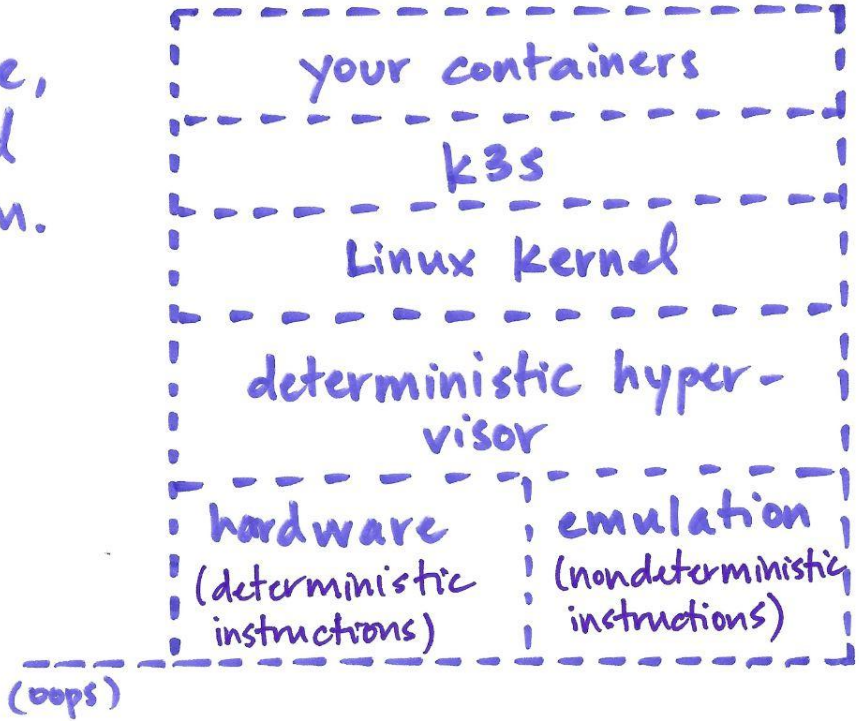
- simple
- fast
- integrated with standard tools

- tedious
- no external dependencies (libraries, DBs, etc.)
- difficult-to-impossible in many languages

# EXTERNAL DETERMINISM : VIRTUAL MACHINES

Somewhere below your code, virtualize the machine and don't allow nondeterminism.

- RewindVM (oss)
- WASM + deterministic WASI shims
- Antithesis



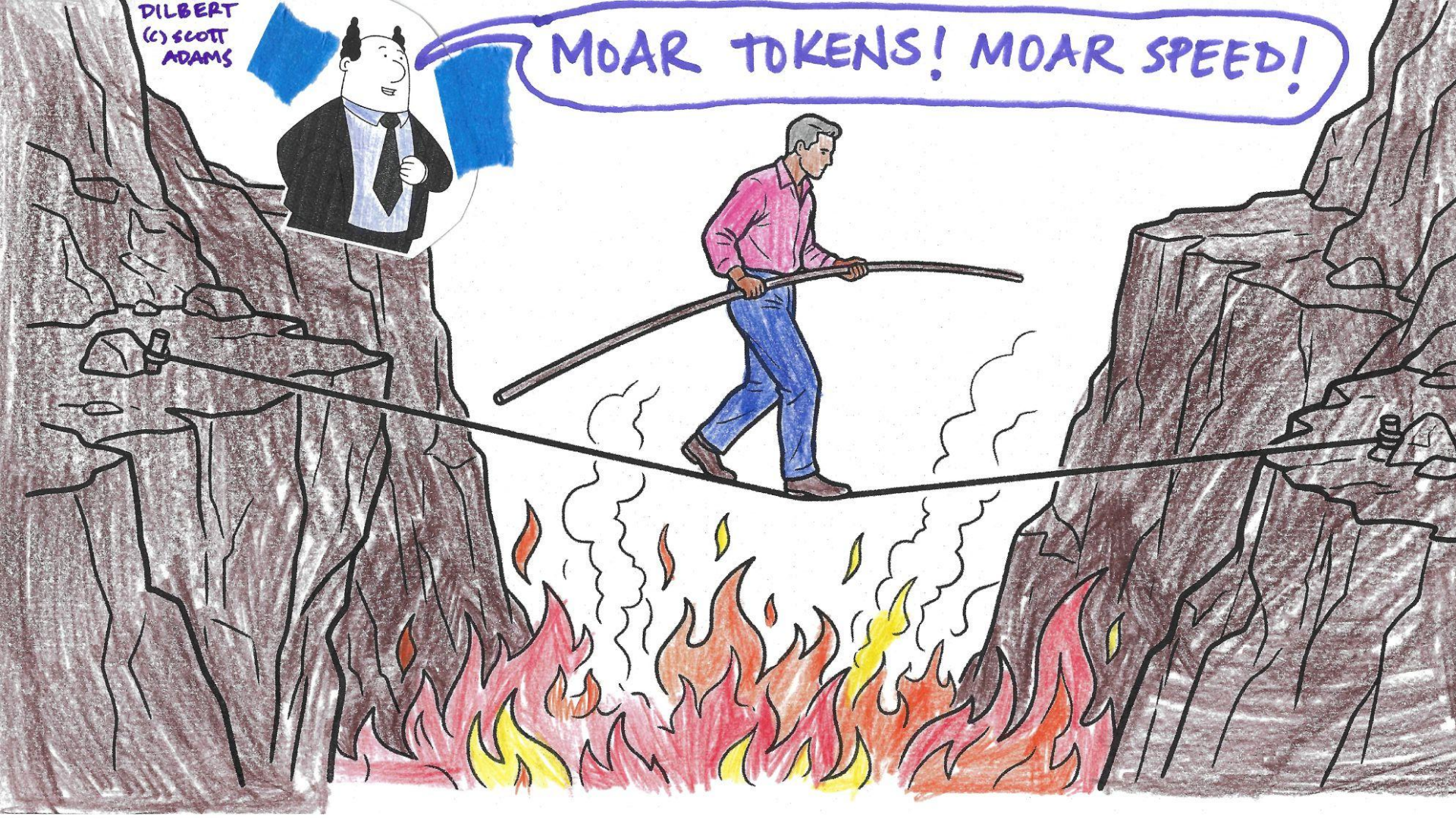
# EXTERNAL DETERMINISM: PROS AND CONS

- any language
- any dependency
- no code changes

- complex
- may require new tools
- few oss options

# DETERMINISTIC SIMULATION TESTING

- DST = PBT + fault injection (like Jepsen) + determinism
- invented independently in the '00s at AWS + FoundationDB
- now used at TigerBeetle, MongoDB, AWS, Confluent, Cloudflare, ...
- most commonly used for database-y systems, but...


DILBERT
(c) SCOTT ADAMS
MOAR TOKENS! MOAR SPEED!

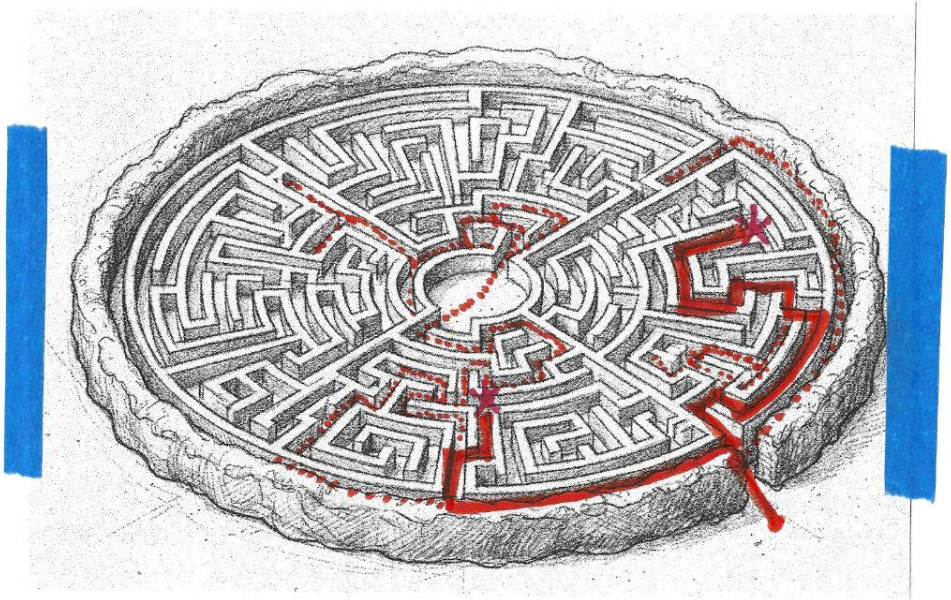
TESTING

unit tests
integration tests
observability
~~chaos testing~~
static analysis
deterministic simulation testing



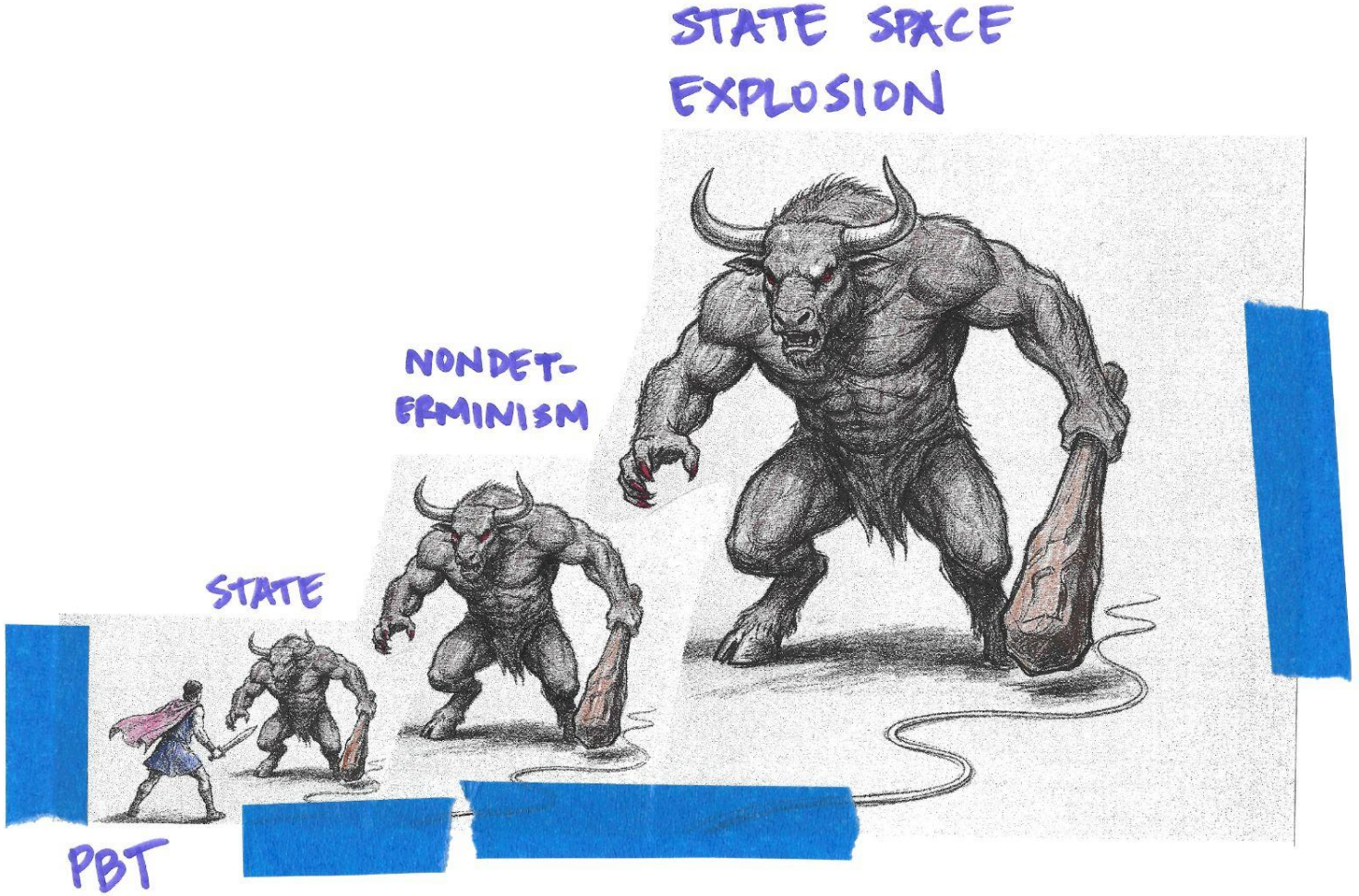
THINKING

architecture
code review
style guides
formal methods
properties



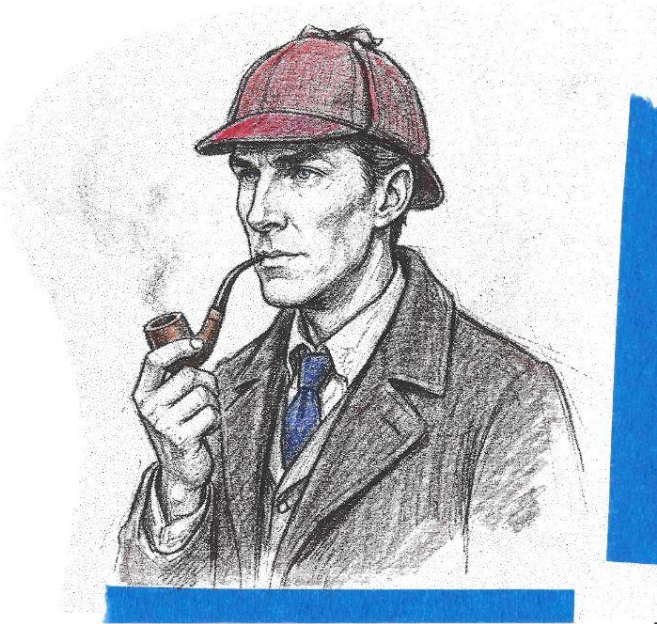
## LLMs ♥ DST

"Look for violations of the properties in PROPERTIES.md. For each suspected violation, write a DST workload to reproduce the bug. Report all confirmed violations."


STATE SPACE EXPLOSION
NONDET-ERMINISM
STATE
PBT

# DST IS A SEARCH PROBLEM

- Real programs have enormous state spaces.
- Random exploration is slow, if it succeeds at all.
- Search strategy matters.
  - coverage
  - swarm testing
  - snapshot pruning
  - others?



# THINKING WHILE TESTING

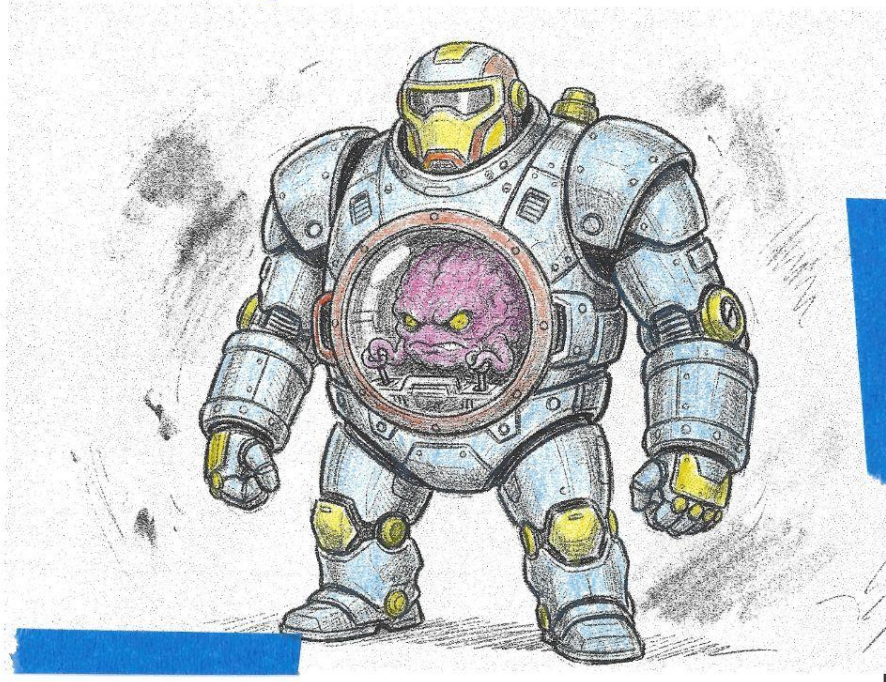
CONCrete + symbOLIC
= CONCOLIC testing

PBT / fuzzing + symbolic logic
to bypass narrow conditionals
("if hash(x) == 0x DEADBEEF")

Rarely used: expensive, struggles
with nondeterminism

Could LLMs help search?

Krang (c) TMNT



## PBT

- QuickCheck
- Hypothesis
- Hegel
- Tyche
- Bombadil

## DST

- Tiger Beetle
- RewindVM
- Turmoil
- Antithesis :

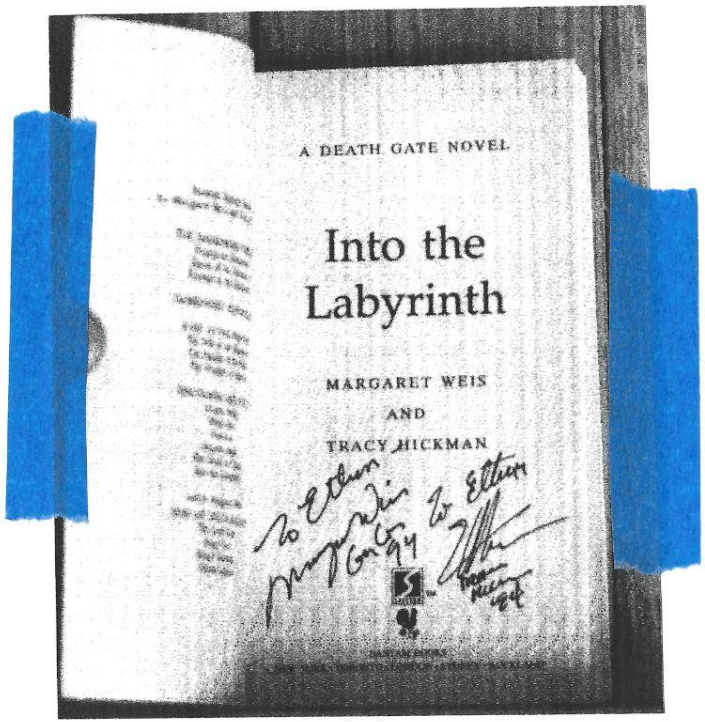


# THANK YOU!



... to Tanya, for inviting me
... to Kyle Kingsbury, for showing me the way
... to my Antithesis colleagues, for their feedback
... to my kids, for help coloring and cutting
... to Gemini, for faux sketches

Come talk DST with me!




# QUESTIONS?

Q: Did you name this talk after book 6 of the Death Gate Cycle?

A: I refuse to admit that I've read it, so no.